Time Protocol Design

- In this implementation of the time protocol messages (requests and answers) are sent as Clear Text (Human Readable)

- The time client supports the following message requests:
  1. GetTime - Used the get the full current data and time.
  2. GetTimeWithoutYear – Used to get the current time only.
  3. GetTimeSinceEpoch – Used to get the current time in seconds since 1/1/1970.
  4. GetClientToServerDelayEstimation – Used to get an estimation of the Delay between client and server. (*will be using 100 messages without waiting each time for a reply)
  5. GetDayAndMonth – Used to get the current day and month only.
  6. GetYear – Used to get the current year only.

- The time server supports the following response answers:
  1. GetTime – Reply the full current data and time in the following format:
     <Day of the week> <Month> <Day> [HH:MM:SS] <Year>
  2. GetTimeWithoutYear – Reply the current time in the following format:
     [HH:MM:SS]
  3. GetTimeSinceEpoch – Reply the current time in seconds since 1/1/1970.
  4. GetClientToServerDelayEstimation – Reply the time stamp on server at the time of the massage arrival (*using GetTickCount**)**
  5. GetDayAndMonth – Reply the current day and month in the following format:
     <Month> <Day>
  6. GetYear – Reply the current year only in the following format:
     [XXXX]

- Each request described in the protocol will be replyed be the correct specific message.
  - All of the request will be sent by a pre-defined string
  - All of the replies will be sent according to the specified formatting

Application Design

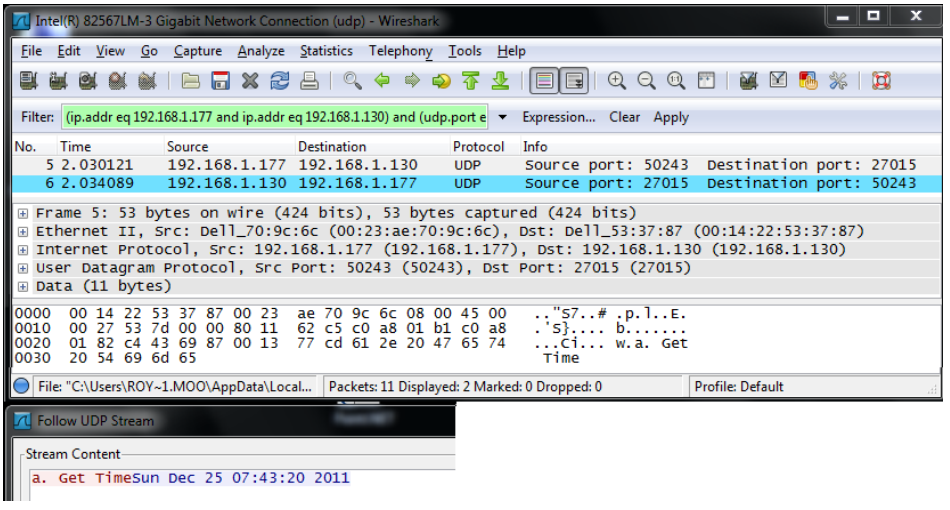
- One application for both client and server, the user selects the desired mode
- The files: "**UDPTime.h**"; "**UDPTime.cpp**"
  Will be Used to include all defines and dependencies needed by both the client and server instances of the application.
- The files: "**UDPTimeClient.h**"; "**UDPClient.cpp**" ; "**UDPTimeServer.h**"; "**UDPServer.cpp**"
  Will be used to operate the desired mode of the application (for client/server instance)

- Client instance:
  1. The user will be prompt to enter The Server IP address (or "local" for 127.0.0.1)
  2. The user will be prompt to enter The desired service (switch)
     This will be maintained by: ***SelectService***
  3. The program will then call the appropriate function to deal with this server
     (Each for every type of services…)
  4. The function will direct the program to create the appropriate message string.
  5. The function will be in charge of sending the right amount of messages and the order they will be sent (*send**receive*…*send**receive* or *send*…*send/receive*…*receive*)
     *In some services, sending and receiving 100 messages will be needed for the client to return the needed information

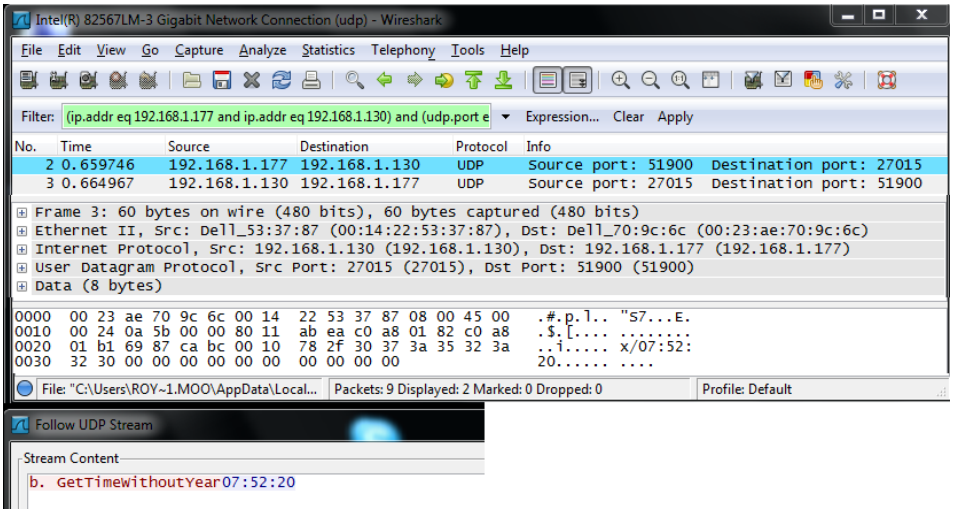
- Server instance:
  1. Server will be running in an endless loop waiting for requests from client.
  2. After a string will be available by the **recvfrom** function it will be parsed by:
     StartServer
  3. An appropriate string will be created by the current function
     (Each for every type of services…)
     *The designated function will be in charge of creating the right type of response string
  4. The created string will be handled by **sendBuff** and sent to the client

Screen captures of different sessions (using Wireshark)

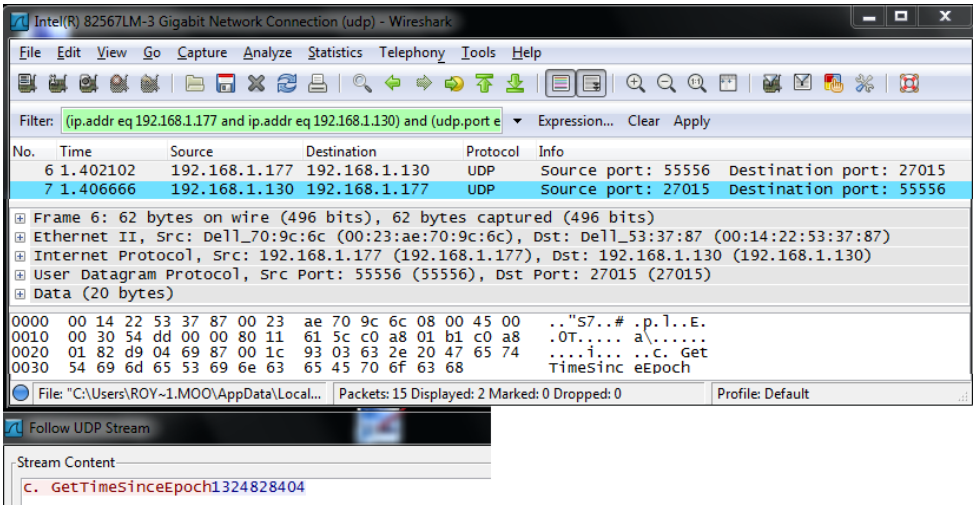
1. GetTime



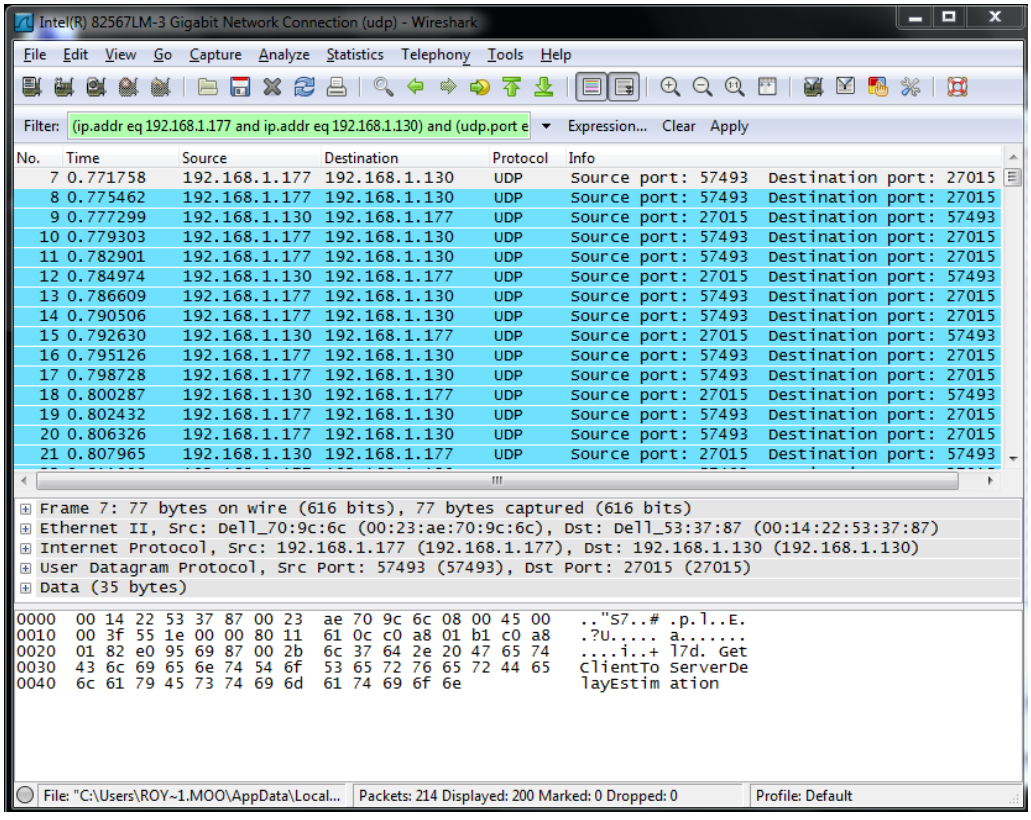
2. GetTimeWithoutYear

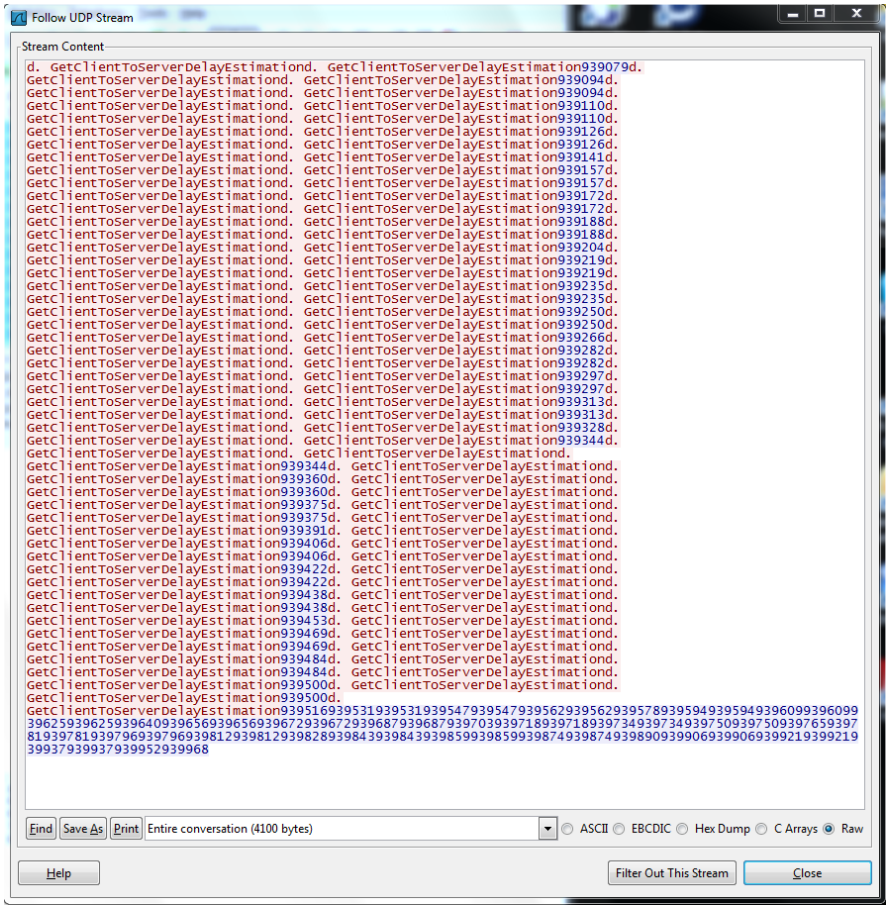


3. GetTimeSinceEpoch

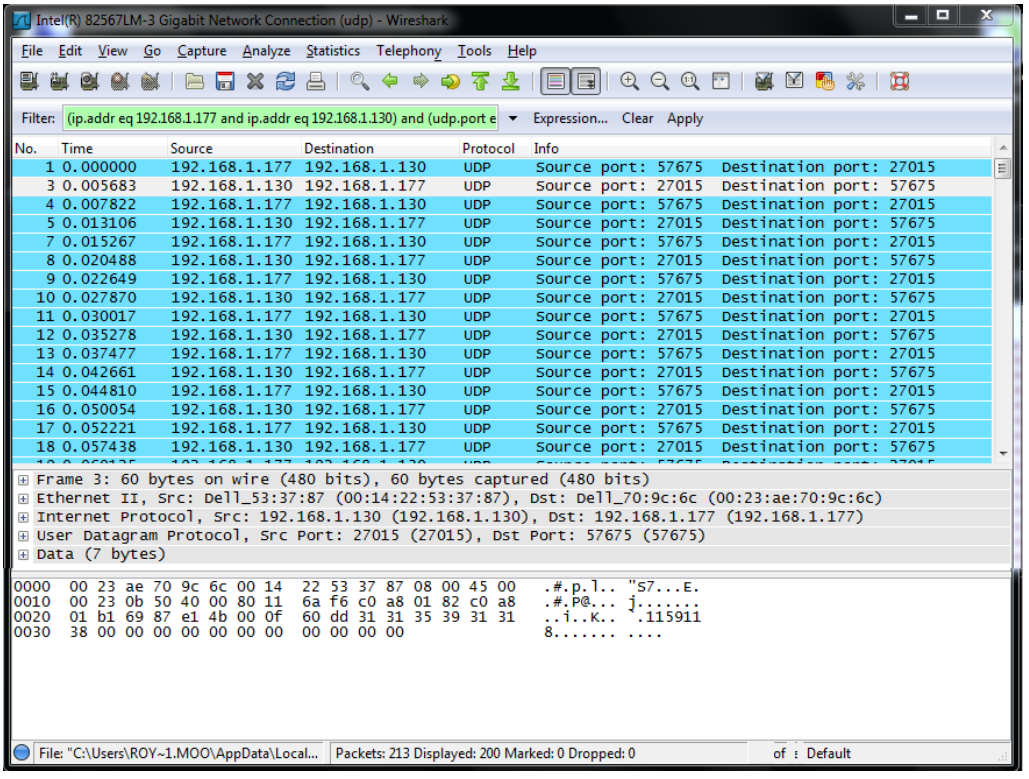


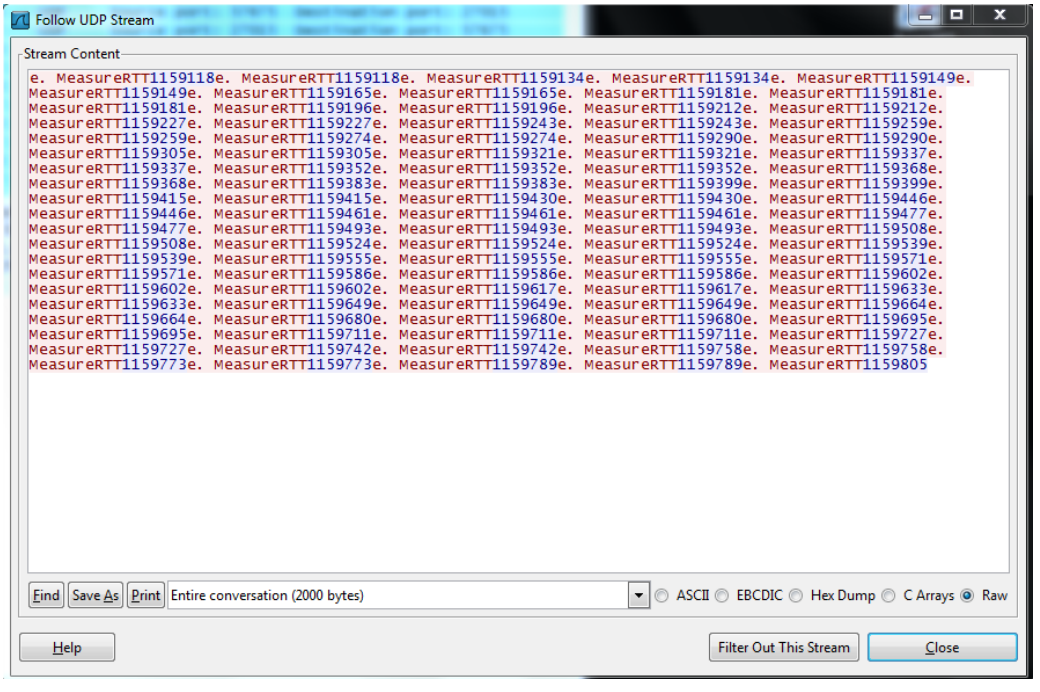
4. GetClientToServerDelayEstimation





5. MeasureRTT





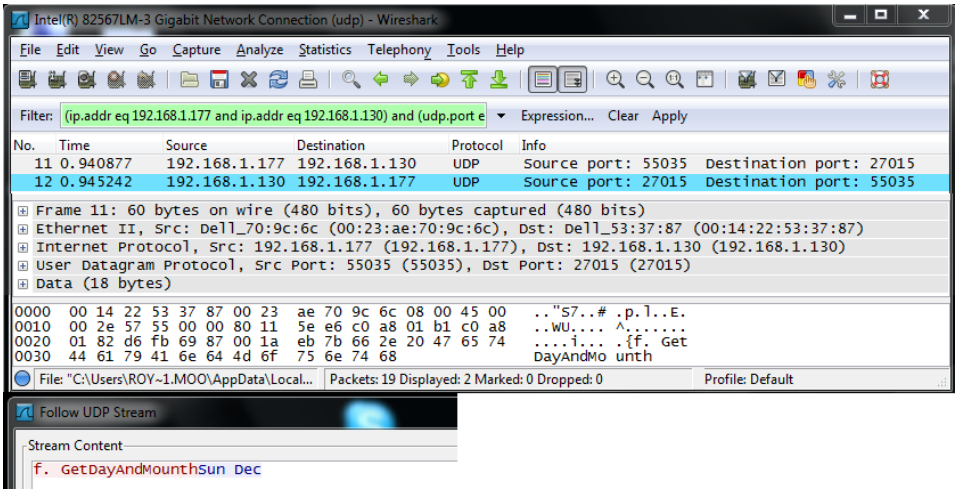
*To calculate to average RTT we used the same request as in 4 (GetClientToServerDelayEstimation)

The request is sent 100 times: send/wait for reply/send/wait for reply…

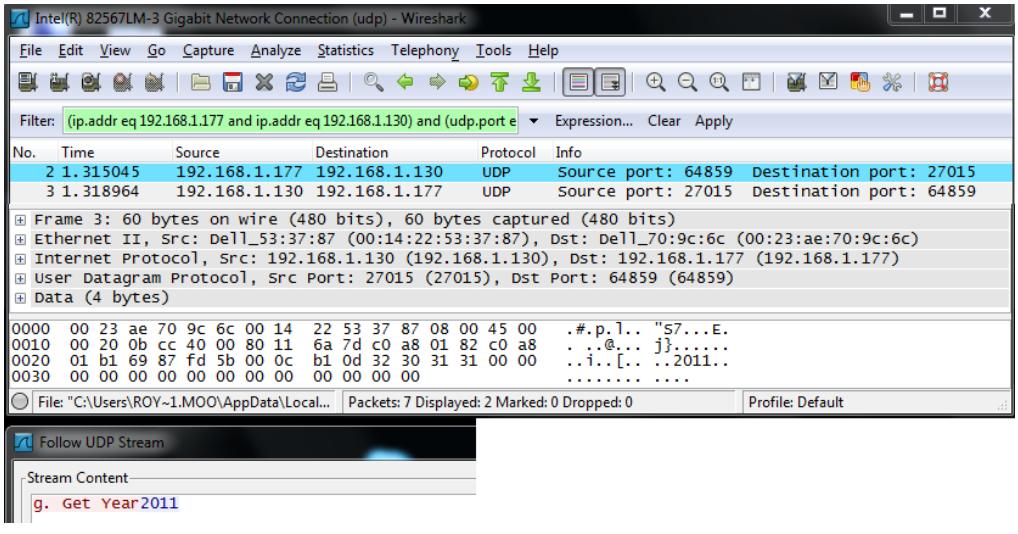
The calculation of the average RTT isn't a part of the original protocol

(It's only a specific utilization of a specific type of request)

6. GetDayAndMonth



7. GetYear



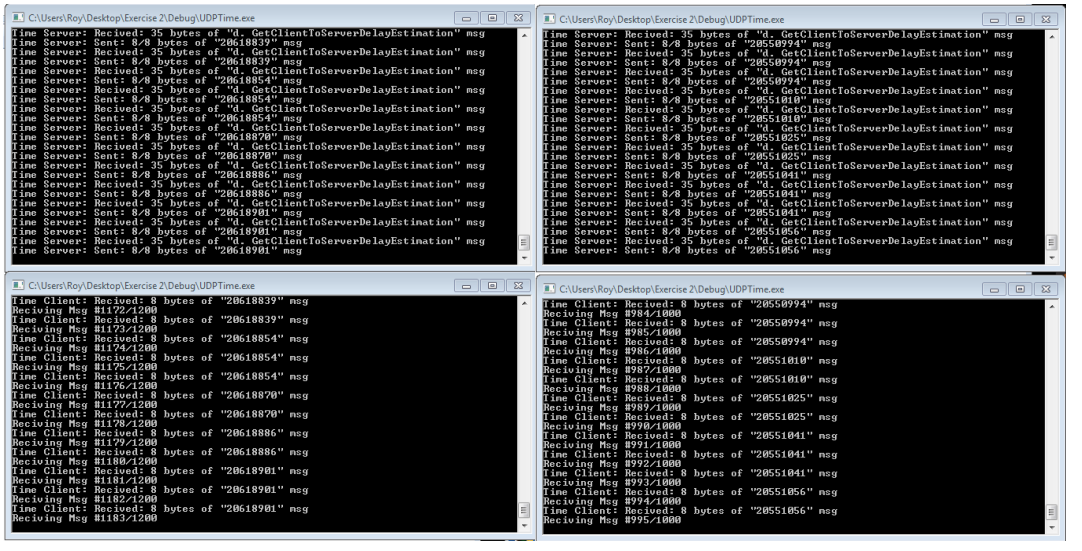
שאלות תיאורטיות

- הריצו את התרגיל ובקשו בקשות GetClientToServerDelayEstimation רבות .
האם ייתכן שהלקוח או השרת "ייתקעו" אם לא ,נמקו .
אם כן ,נמקו והציעו תיקון בתוכנית שלכם בכדי שהלקוח והשרת לא "יתקעו."

- מבחינת ההשהיות השונות שאנו כבר מכירים(אותן 4 השהיות שנלמדו בכיתה), מה מבטא הגודל הממוצע אותו הלקוח מחשב על סמך תשובות השרת בבקשת GetClientToServerDelayEstimation ?
(המלצה :נתחו גודל זה מבחינה תאורטית ולאו דווקא על סמך התוצאות המתקבלות מההרצה בפועל שעלולות להטעות ,בפרט כאשר מבוצעות על אותו המחשב)

- יתכן בהחלט כי גם השרת וגם הלקוח יתקעו.
הדבר נובע מתצורת העבודה אותה בחרנו ליישם בתרגיל : blocking
בהחלט יתכן כי באופן תיאורטי כל הבקשות של הלקוח יאבדו בדרך ואז השרת ימתין\יתקע עד לקבלת המידע האבוד... (התיאור זהה גם לגבי הלקוח)

בדיקה בפועל מוכיחה כי אכן יש תקיעות:



הפתרון המתבקש לבעיה הוא לשנות את תצורת העבודה של ה socket
לתצורת non-blocking (כלומר יש דגימה של החוצץ בכל פרק זמן כדי לקרוא ונמנע המצב של התקיעה)

- ההשהיה שאנו מודדים בסעיף Estimated Delay מבטאת את Propagation Delay על הקשר שבין השרת ללקוח, דרך נוספת להסתכל על זה (כמו שלמדנו) היא כעל הזמן שלוקח למידע לעבור ב"צינור" מצד בלקוח לצד השרת.